LSEG

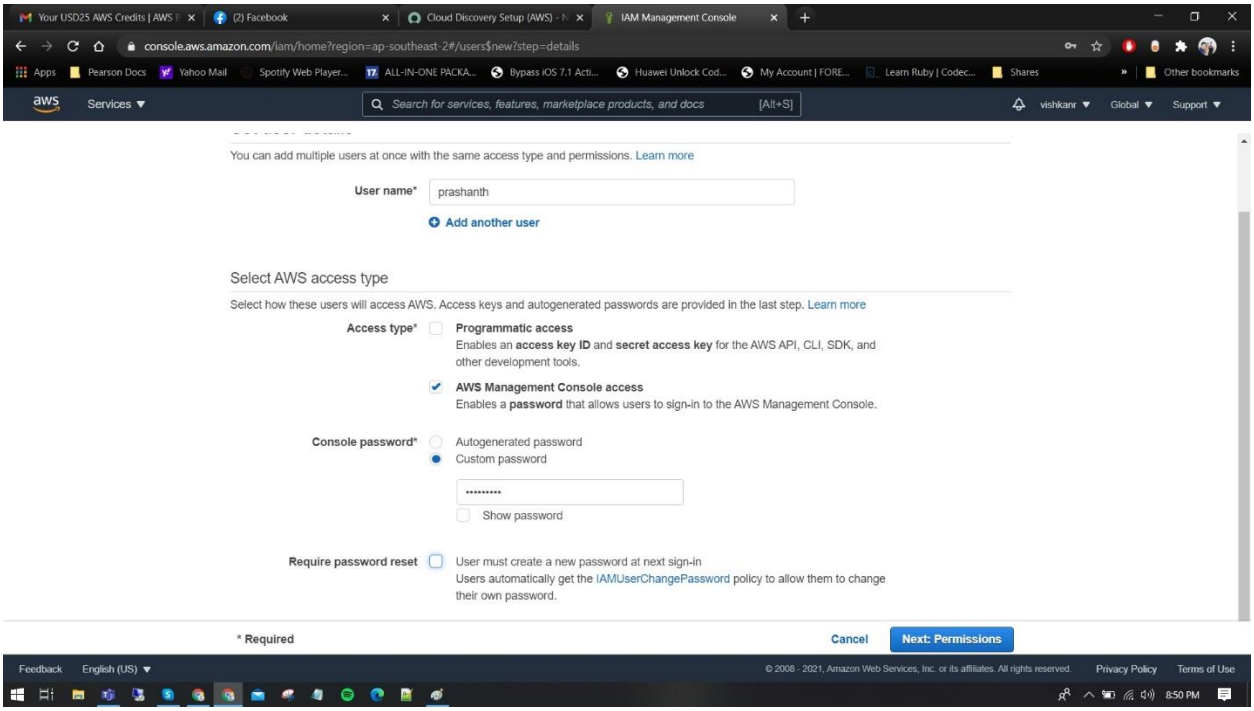
Capital Markets

Ratnagopal, Vishkan

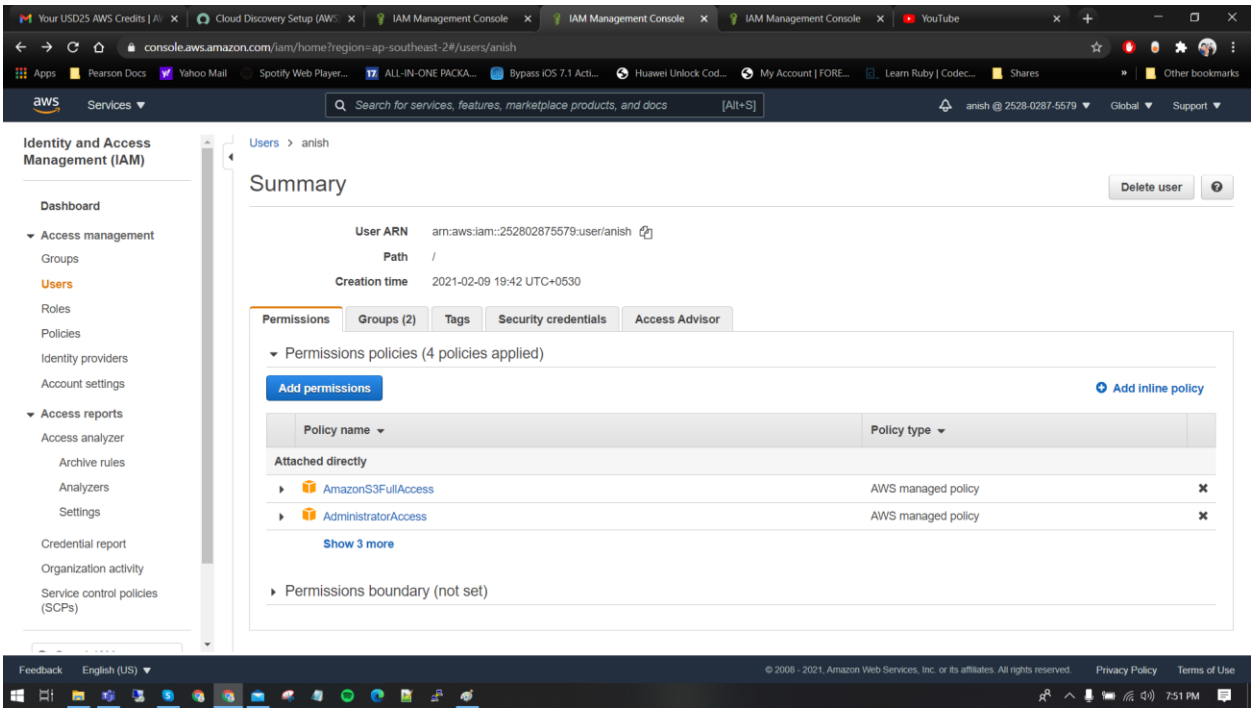
Contents

Capital Markets Application Support/DevOps

Creating IAM user -

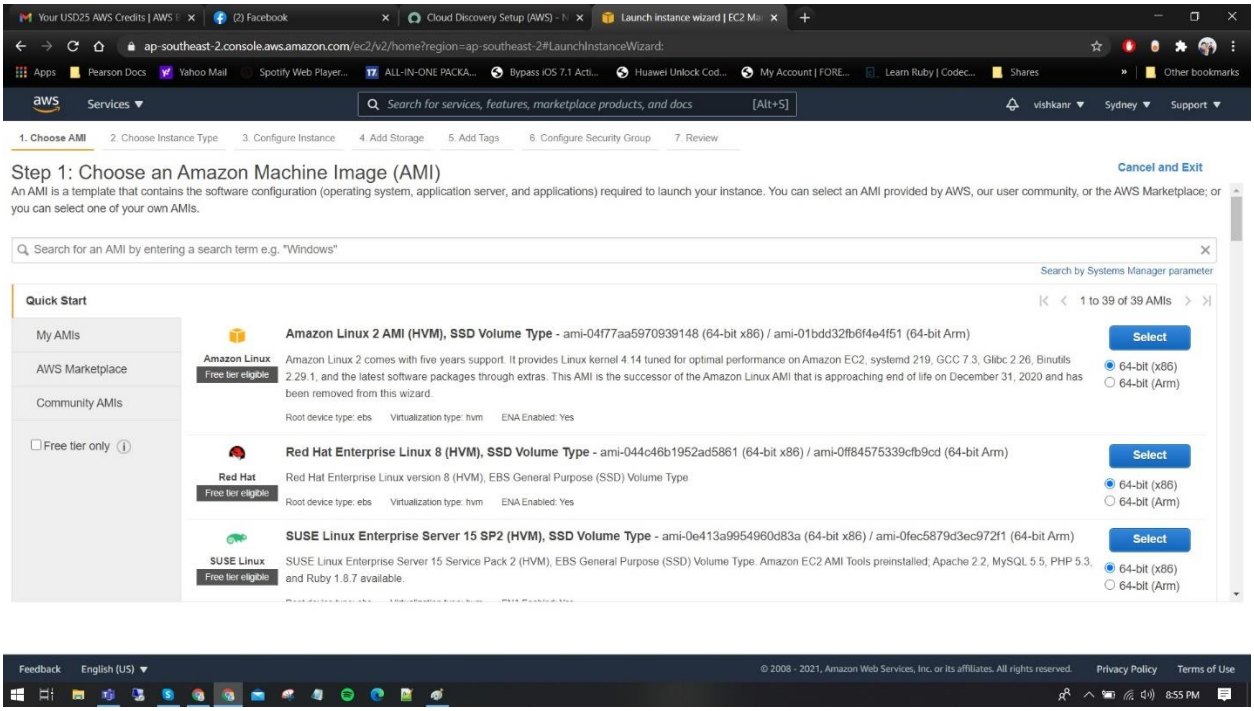


Creating user with AWS management console access and S3 bucket.

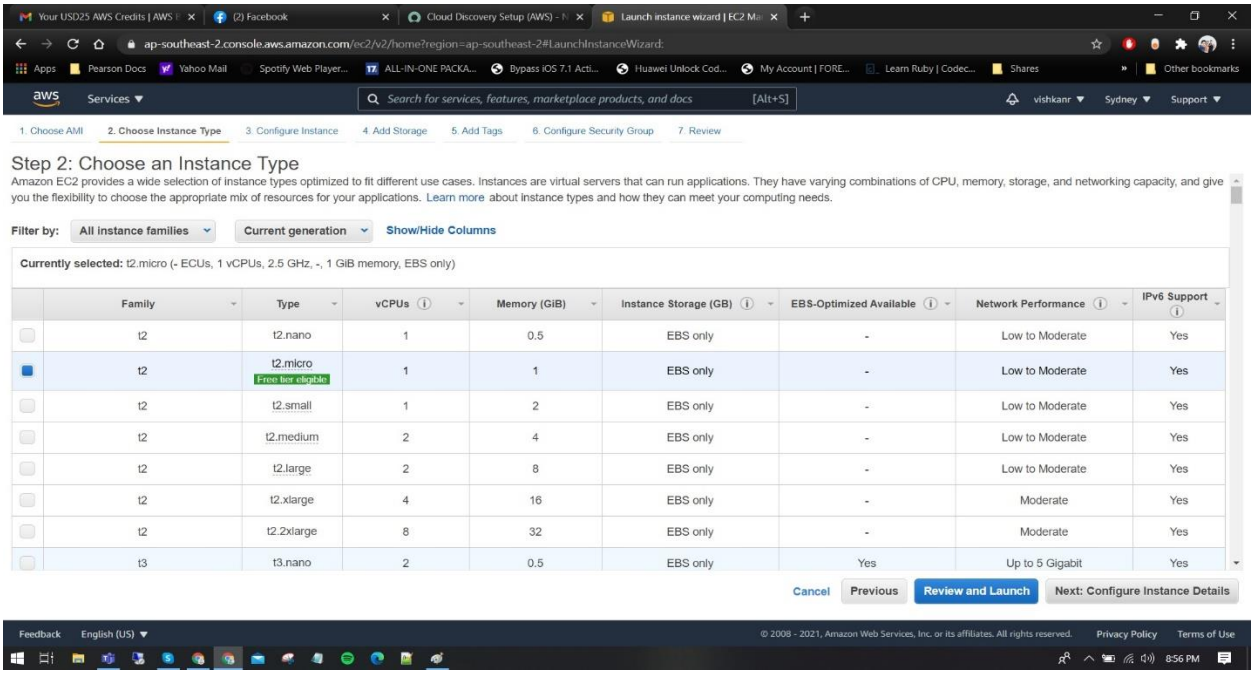


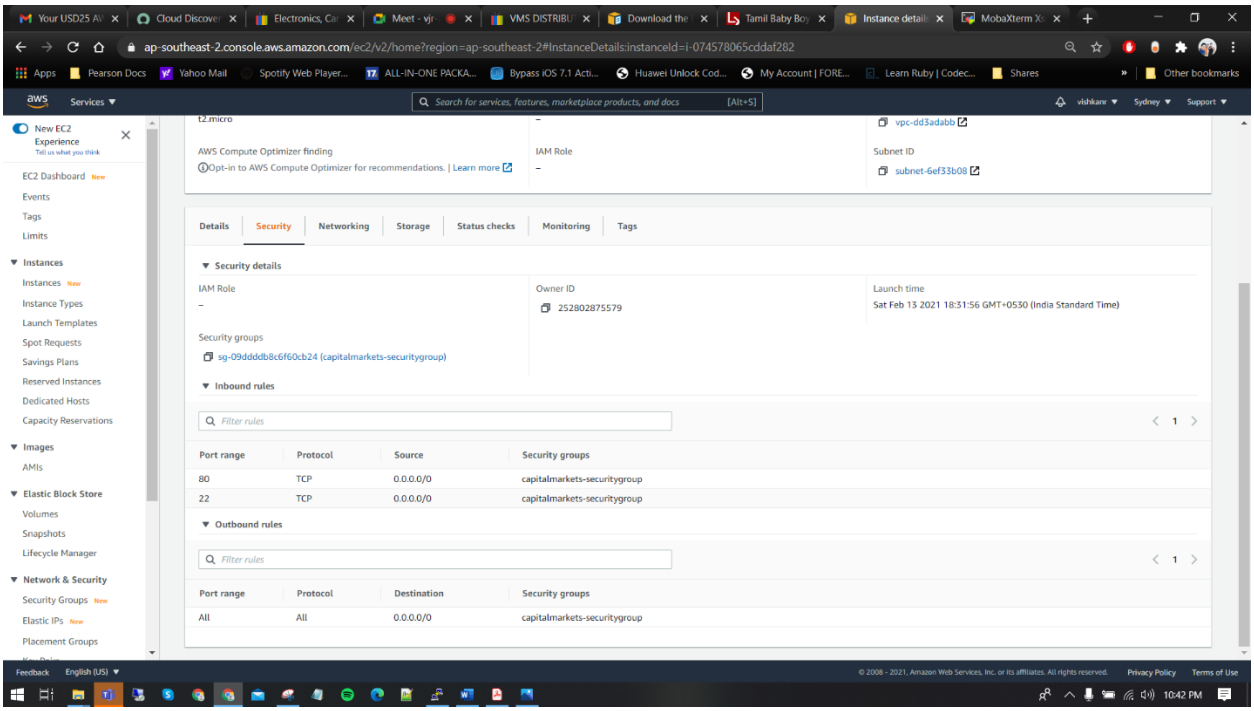
2 roles were created one for general user with read access and other role for admin access this user is provided with admin access for full control.

Creating EC2 Instance -



Creating an ec2 instance to host Apache server. Selected red hat enterprise Linux 8 with default configurations as shown below.





Security group was created. Inbound and outbound rules were set to publicly accessible to SSH and access the web server as well.

Setting up Apache Webserver -

Following commands are used to set up the web server.

Refresh repository:

sudo yum update -y

Install php version 7.2, run:

sudo yum install php -y

Start PHP fpm service:

sudo systemctl start php72-php-fpm.service

Apache HTTP Server, Install the httpd package:

yum install httpd

Enable and start the httpd service:

systemctl enable --now httpd

change the html content:

echo "Capital Markets" > /var/www/html/index.html

Install Python3:

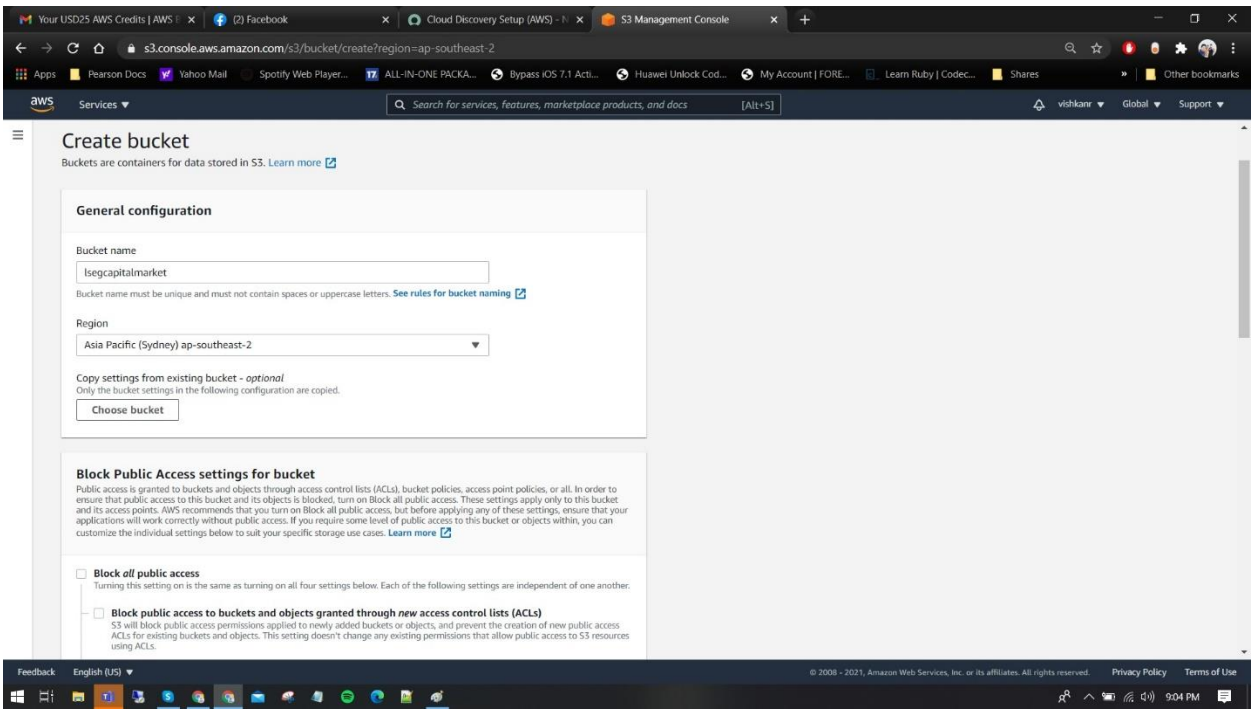
sudo yum -y install python3-pip

external Import library:

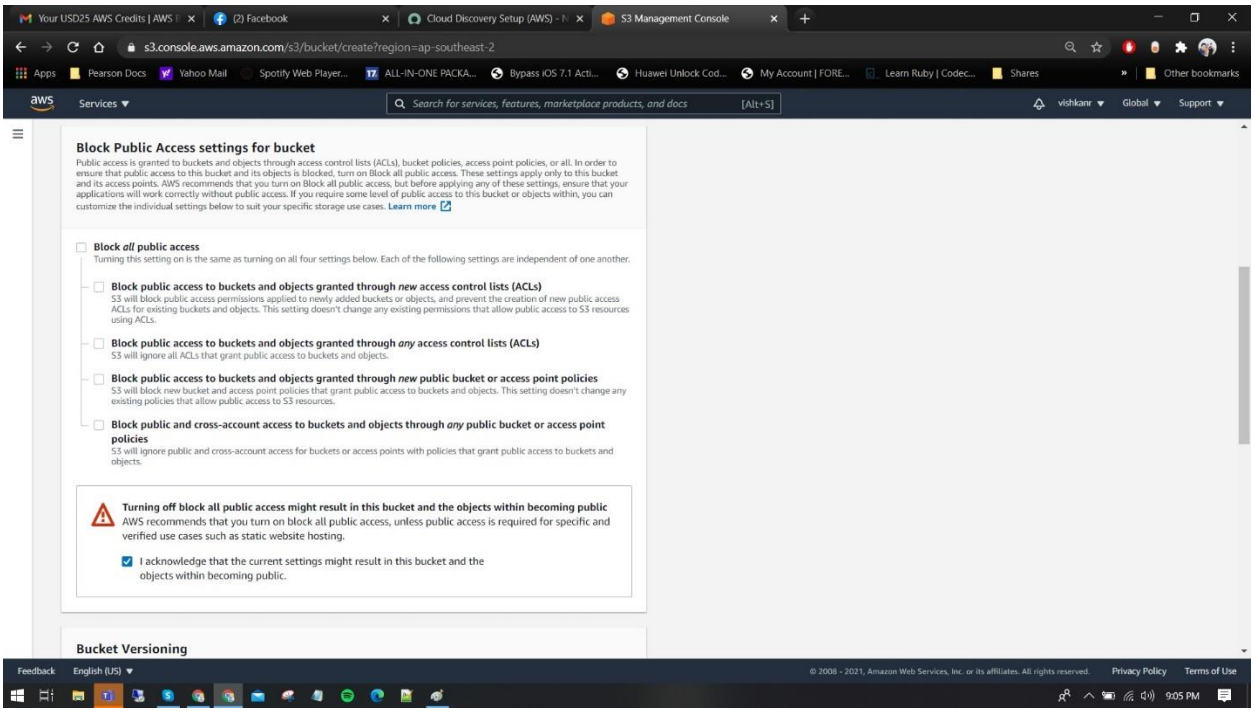
import paramiko

import boto3

Creating S3 Bucket -



S3 bucket was created to store log information of the webserver results.



Bucket was given public access to view the log information.

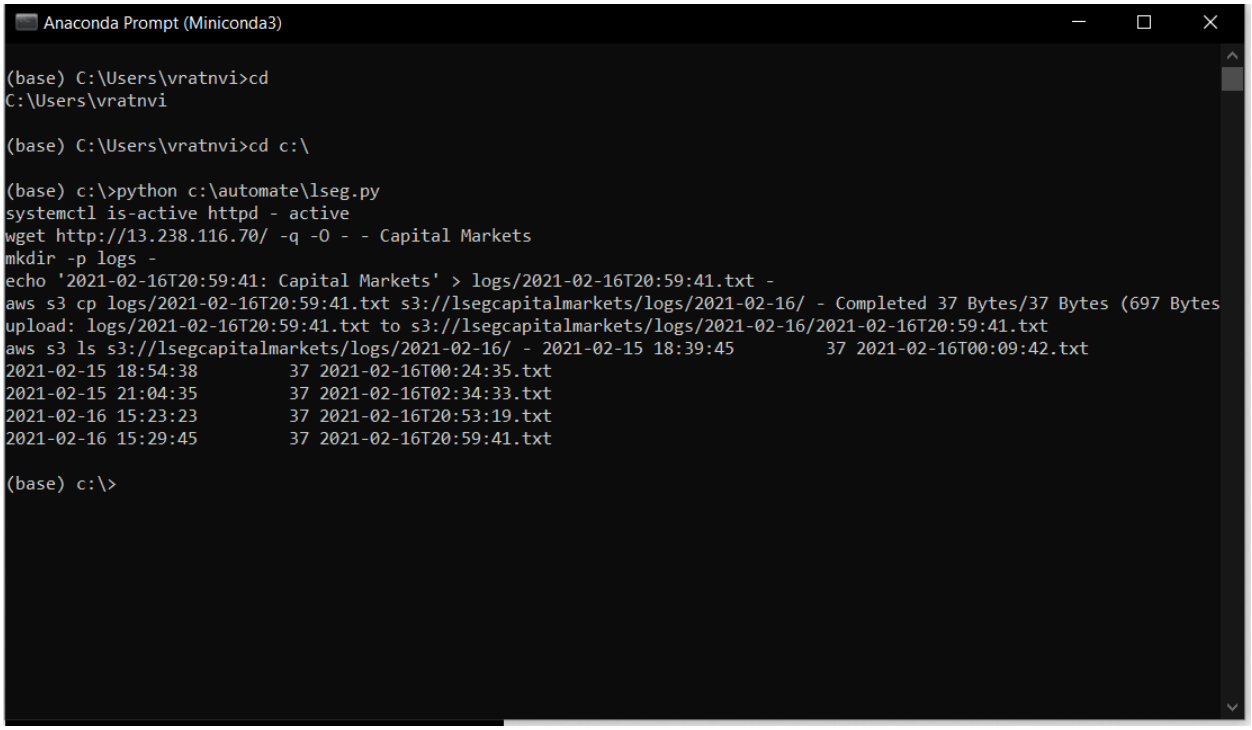
Writing the script-

Following script will test the status of the web server if it's running, script will pass the result and creates a folder in S3 bucket under the following path S3://lsegcapitalmarkets/logs/<date>/ Logs will be created as a text file.

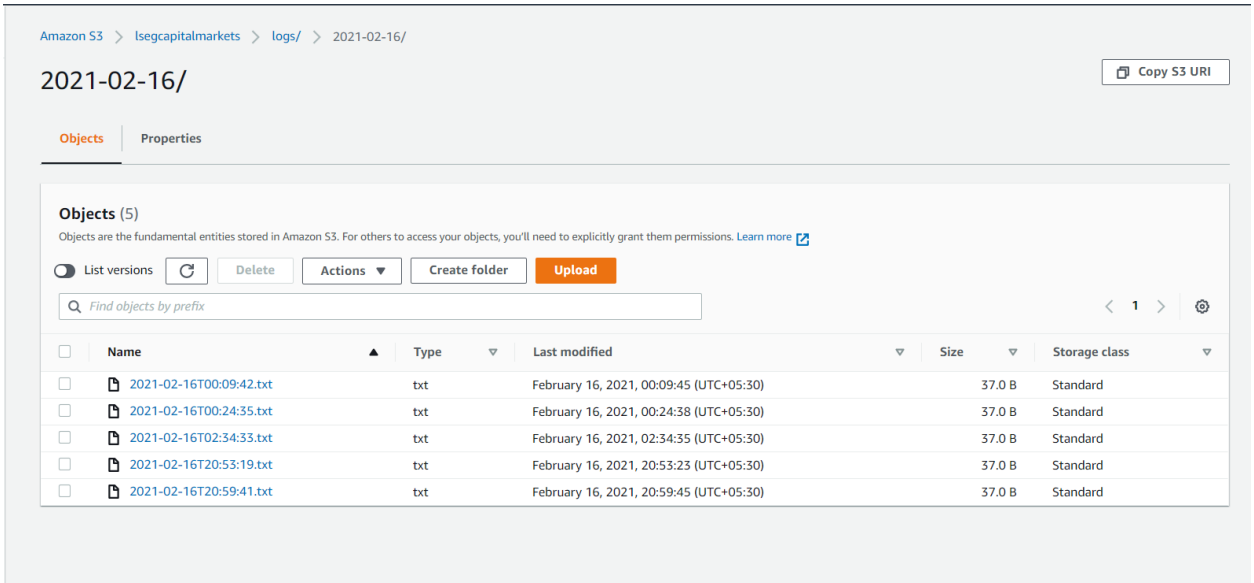
```
#importing libriary
import paramiko
import datetime


def main():
    # set up environmental variables
    host_name = "13.238.116.70"
    user_name = "ec2-user"
    key_path = "C:/automate/lseg_capital_markets_pem"
    service_name = "httpd"

    # get ssh client object
    ssh_connection = paramiko.SSHClient()

    # remove known host error
    ssh_connection.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    # create connection to host
    ssh_connection.connect(
        hostname=host_name,
        username=user_name,
        key_filename=key_path
    )

    # set up bash commands in string variables
    cmd_service_status = f"systemctl is-active {service_name}"
    cmd_start_httpd = f"sudo systemctl start {service_name}"
    cmd_read_content = f"wget http://{host_name}/ -q -O -"

    # set up date/time variables
    timestamp = datetime.datetime.now().strftime("%Y-%m-%dT%H:%M:%S")
    date = datetime.datetime.now().strftime("%Y-%m-%d")

    # check httpd status
    httpd_status = execute(ssh_connection, cmd_service_status)

    # if httpd is not active then start the service
    if httpd_status != "active":
        execute(ssh_connection, cmd_start_httpd)

    # read web content from the hosted httpd web page
    web_content = execute(ssh_connection, cmd_read_content)

    # write to file and sync to s3
    write_log(ssh_connection, web_content, timestamp, date)

    # close ssh connection
    ssh_connection.close()
```

```python
def sync_s3(connection, file_location, date):
    # location of the s3 bucket
    bucket_location = "s3://lsegcapitalmarkets/logs/"

    # command variable for s3 sync
    cmd_copy_bucket = f"aws s3 cp {file_location} {bucket_location}{date}/"
    cmd_verify_bubcket = f"aws s3 ls {bucket_location}{date}/"

    # sync file with s3
    execute(connection, cmd_copy_bucket)

    # verify if files are copied
    execute(connection, cmd_verify_bubcket)


def write_log(connection, content, timestamp, date):
    # text file location to save on local
    file_location = f"logs/{timestamp}.txt"

    # command variables for file write
    cmd_mkdir_log = "mkdir -p logs"
    cmd_write_file = f"echo '{timestamp}: {content}' > {file_location}"

    # create log folder if not exists
    execute(connection, cmd_mkdir_log)

    # write to file
    execute(connection, cmd_write_file)

    # sync file to s3
    sync_s3(connection, file_location, date)


def execute(connection, command) -> str:
    # execute script
    std_in, std_out, std_err = connection.exec_command(command)

    # result of execution from shell standard out
    result = std_out.read().decode().strip()

    # print all outputs
    print(f"{command} - {result}")

    # return result status from script
    return result


if __name__ == '__main__':
    main()
```

Results –



By running the first python script it will check the status of the web server (httpd). Then the results echo the date, time and content that was provided in the web server. As an additional effort the log file is created and passed to the S3 bucket for tracking status of the webserver.



Bellow script will be running externally to the S3 and creates a compressed file according to date of the logs locally and uploads back to S3 as a tar file and send a mail notification if there are no download files for the specified date.

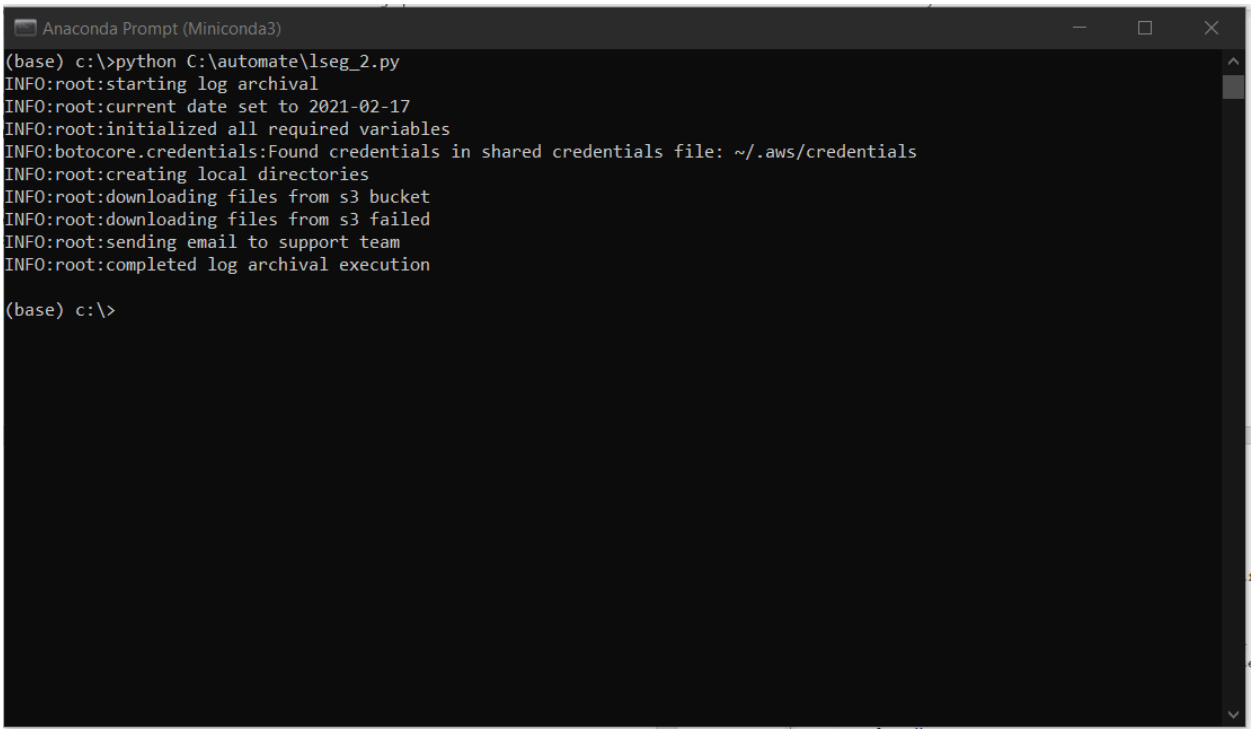
```
import os
import boto3
import datetime
import shutil
import glob
import tarfile
import smtplib
import logging
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText


def main():
    logging.info("starting log archival")

    # current date
    current_date = datetime.datetime.now().strftime("%Y-%m-%d")

    logging.info(f"current date set to {current_date}")

    # s3 variables setup
    resource_name = "s3"
    bucket_name = f"lsegcapitalmarkets"
    archive_url = f"logs_archive/{current_date}"

    # artifact variables setup
    logs_dir = "logs_raw"
    dir_tree = f"{logs_dir}/{current_date}"
    artifact_name = f"{current_date}.tar.gz"

    # mailing variables setup
    username = "vishkanr@gmail.com"
    password = "Civic1999"
    recipient = "vishkanraj@yahoo.com"
    subject = "FAILURE - Logging archive process failed"
    body = f"The log archival process has failed on {current_date} due to "

    logging.info("initialized all required variables")

    # get s3 bucket
    s3 = boto3.resource(resource_name)
    bucket = s3.Bucket(bucket_name)

    logging.info("creating local directories")

    # if local path doesnt exist create it
    if not os.path.exists(dir_tree):
        os.makedirs(dir_tree)
```

```
    # keep count of downloaded files
    downloaded_count = 0

    logging.info("downloading files from s3 bucket")

    # iterate all keys
    for obj in bucket.objects.all():
        # filename to download taken from the key
        filename = str(obj.key.rsplit('/')[-1]).replace(':', '')

        # download only what is available for specified date
        if current_date in filename:
            bucket.download_file(obj.key, f"{dir_tree}/{filename}")
            downloaded_count += 1

    # if download count is greater than 0 upload to s3, else send email to support team
    if downloaded_count > 0:
        logging.info("downloading from s3 successful")

        # create tar artifact from all files downloaded
        create_artifact(dir_tree, "txt", artifact_name)

        logging.info("uploading tar archive to s3")

        # upload zip artifact to s3 bucket
        s3.meta.client.upload_file(artifact_name, bucket_name, f"{archive_url}/{artifact_name}")

        logging.info("cleaning directories")

        # remove tar file
        os.remove(f"{artifact_name}")

        # remove raw logs
        shutil.rmtree(logs_dir)
    else:
        logging.info("downloading files from s3 failed")

        # update error
        body += "no files being downloaded for set date. Please verify urgently!"

        logging.info("sending email to support team")

        # send email
        generate_email(username, password, recipient, subject, body)

    logging.info("completed log archival execution")
```

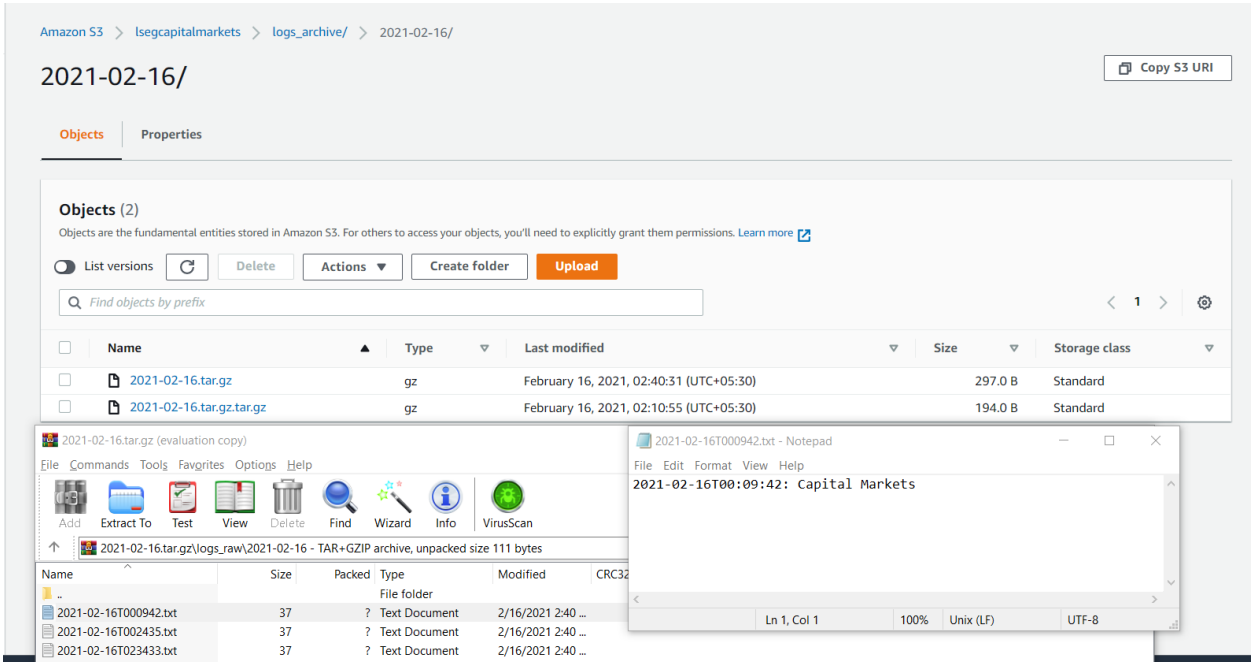
```
def create_artifact(source_dir, file_extension, artifact_name):
    # open tar creation
    tar = tarfile.open(artifact_name, "w:gz")

    # iterate through source destination looking for specified file extension
    for path in glob.glob(f"{source_dir}/*.{file_extension}"):
        tar.add(path)

    # close tar creation
    tar.close()


def generate_email(username, password, recipient, subject, body):
    # Setup email as multipart MIME
    mail = MIMEMultipart()

    # set email structure
    mail['From'] = username
    mail['To'] = recipient
    mail['Subject'] = subject
    mail.attach(MIMEText(body, 'plain'))

    # setup SMTP session with gmail port and enable security options
    session = smtplib.SMTP('smtp.gmail.com', 587)
    session.starttls()

    # login to email account and send email
    session.login(username, password)
    text = mail.as_string()
    session.sendmail(username, recipient, text)

    # close smtp session
    session.quit()


if __name__ == '__main__':
    # set logging level to info
    logging.basicConfig(level=logging.INFO)

    # start main execution
    main()
```

The script is saved in the following path C:\automate with the “pem” file of the ec2 instance. The region, access key and secret access key are stored in following path for the script to access C:\Users\vratnvi\.aws.

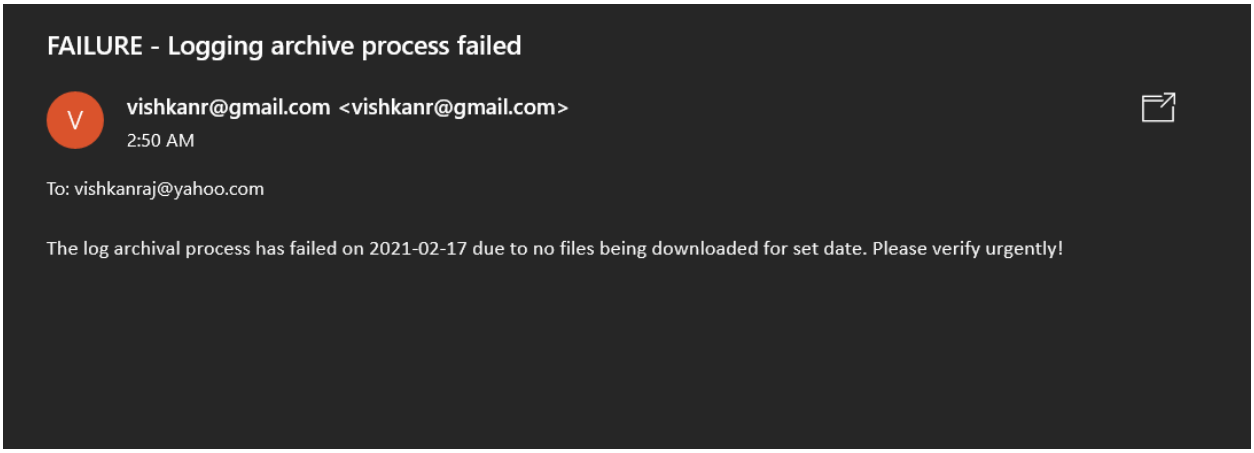
Results –



By running the 2nd script, it will create a compressed file within that log file is created with time stamp and content. The log file will be downloaded to local PC (underdevelopment) and a mail is if there are no download files for the specified date.

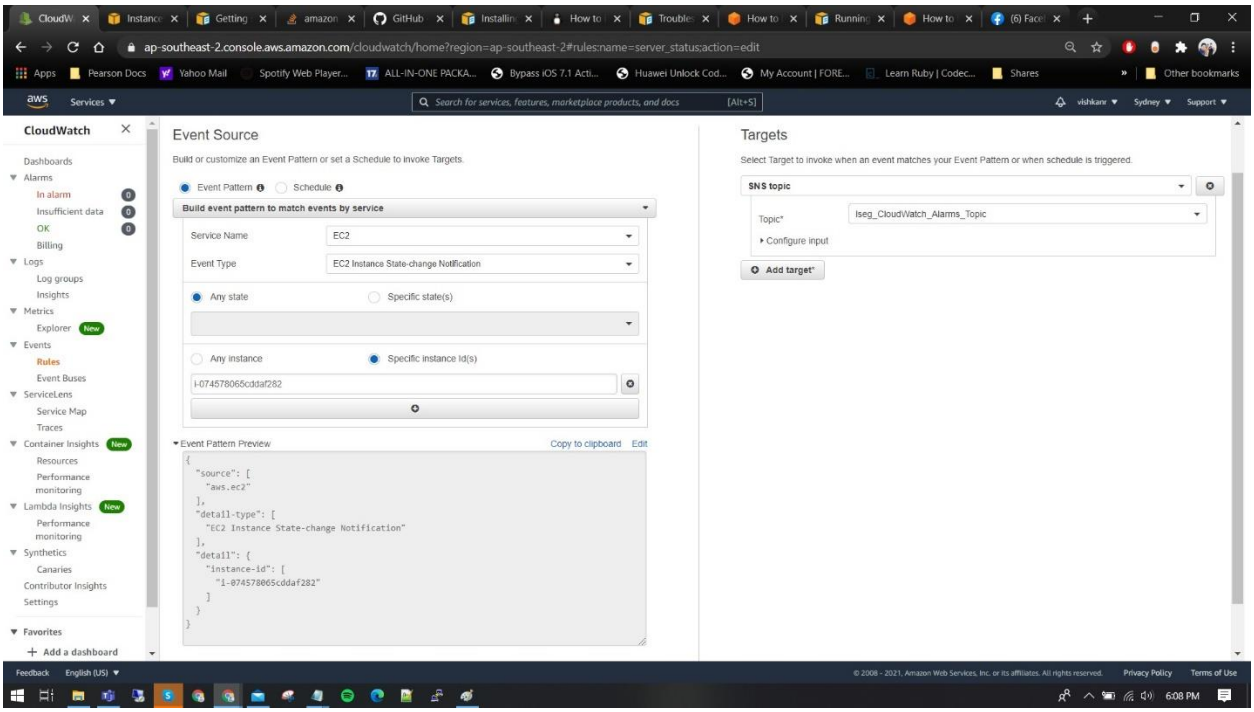


Bellow shows the failure if there are no download files for the specified date.



Additional Task -

As an additional task cloud watch was set up if the server goes down or restarts an event is triggered.



SNS was configured to notify the stake holders.